Write a Basic Object Tracking Drone Application

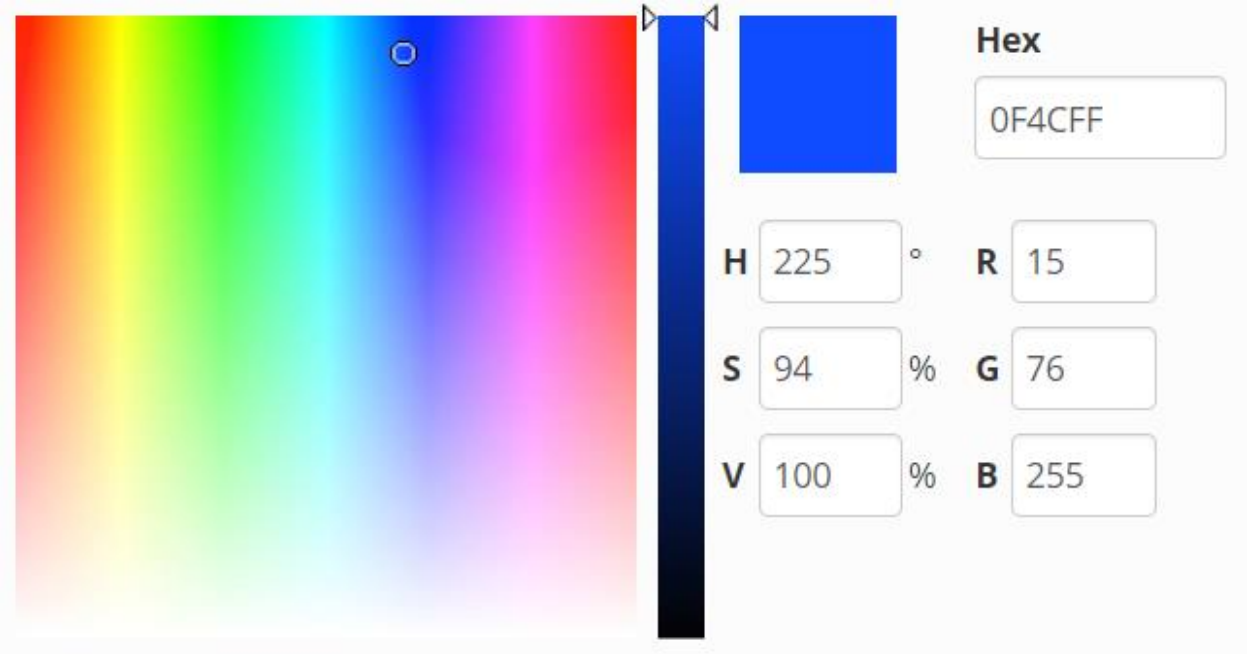
With OpenCV, MAVSDK and PX4 SITL Simulation

APPENDIX

COLOR SPACES

A color space is a mathematical model by which we can describe colors as tuples of numbers, typically 3 or 4 [Reference 2]. Figure 1 shows a color example in RGB (or BGR) and HSV color spaces.

In BGR color space each individual color is represented by the amount of Blue, Green and Red components. To form the given color, each channel takes a value in the range from 0 to 255. For instance, the color in the example is represented by the BGR tuple (255, 76, 15).

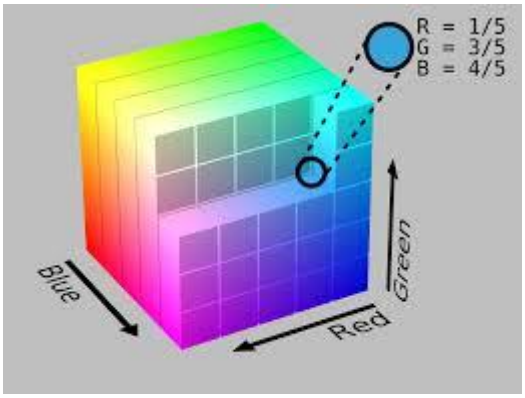


https://alloyui.com/examples/color-picker/hsv.html

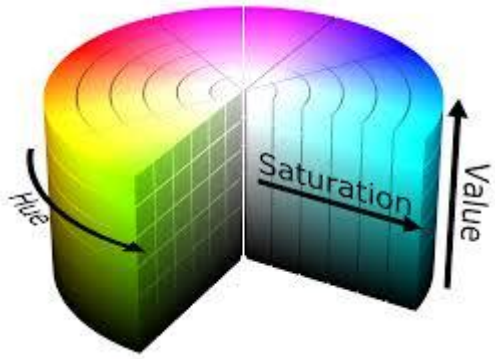
Figure 1

On the other hand, the HSV color space describes each individual color by combining different proportions of Hue, Saturation and Value. 'Hue' models the color type and refers to the primary and secondary colors and any combination of them; as if it is reddish, bluish, greenish, etc. It is the wavelength of the color. 'Saturation' represents the amount of white for a given Hue that lightens it; if the color is fully saturated, the amount if white is zero. By adding white to the selected Hue in various amounts, we obtain paler "tints" of that color or Hue [Reference 3]. Finally, 'Value' describes the brightness or intensity of the color. Lower Values makes a given color tint (that is, a combination of a given Hue and Saturation) brighter. Higher Values add more

gray (or a certain amount of black) to the color to create different tones. The color example above is defined by the HSV tuple (255, 94, 100). Note that in Figure 1, Hue is defined in a range between 0~360 degrees, Saturation and Value between 0~100%. In OpenCV however, Hue is defined in a range between 0~179, Saturation and Value between 0~255.



RGB Color Space



HSV Color Space

LOW-PASS MOVING AVERAGE FILTER

A Low-Pass Moving Average filter takes the last measurement (in our case, the measurement will be the target's [width, height] coordinates) and averages it along with a number of the last previous measurements, to obtain the current filtered value. For instance, if it is a '20 point' moving average filter, it will take the current measurement along with the last 19 previous measurements, and compute the average; which will be the filtered value of the current measurement. So, a moving average filter is in principle just an average calculator that takes '**N-1**' previous measurements $\mathbf{In_1}$ to $\mathbf{In_{N-1}}$, along with the current measurement $\mathbf{In_N}$, and provides a filtered output $\mathbf{Out_N}$, as an average of the last **N** inputs. For the next measurement the same procedure will be repeated, and so on, so forth.

Equations (7) and (8) describe mathematically this type of filter [Reference 3], which is technically a Finite Impulse Response (FIR) Low-Pass filter.

$$y[i] = \frac{1}{N}\sum_{j=0}^{N-1} x[i+j] \quad (7)$$

$$y[21] = \frac{x[21]+x[22]+x[23]+x[24]+x[25]}{5} \quad (8)$$

If you recall, in the 'main' function, after we obtained the detected object's centroid, we applied the filter to the centroid coordinates. Code Listing 3 shows the code for the **moving_average_filter(coord**) function, which receives as argument the (width, height) coordinates in a 'coord' Python dictionary. In lines 4-5, we append the 'width' coordinate to the **filt_buffer['width']** list, and the 'height' coordinate to the **filt_buffer['height']** list. In lines 7-9 we check if the **filt_buffer['width']** list has more than **NUM_FILT_POINTS** elements (**filt_buffer['height']** list should have the same number of elements); which is the number of points for the filtering we want to apply (20 in our example). If so, we pop from both lists the oldest element, to always have no more than **NUM_FILT_POINTS** elements in both lists.

In line 11, we get the number of elements for the 'width' list (it will be the same for the 'height' list), then in lines 13-17, we calculate the averages by first adding all elements in each buffer and then, dividing the sums by '**N**'. We round the results and convert them to int, because these values are pixel coordinates in the "camera image" coordinate system, which always have to be integers. Finally, in line 19 we return both values in a dictionary.

REFERENCES

[Reference 2] http://www.arcsoft.com/topics/photostudio-darkroom/what-is-color-space.html

[Reference 3] https://color-wheel-artist.com/hue/

[Reference 3] https://dspguide.com/ch15.htm